



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
-----------------	-------------	----------------------	---------------------	------------------

09/547,288

04/11/2000

Nir N. Shavit

1004-4663

4871

42714

7590

05/05/2006

ZAGORIN O'BRIEN GRAHAM LLP (004)
7600B NORTH CAPITAL OF TEXAS HIGHWAY
SUITE 350
AUSTIN, TX 78731-1191

EXAMINER

LI, AIMEE J

ART UNIT

PAPER NUMBER

2183

DATE MAILED: 05/05/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary

Application No.

09/547,288

Applicant(s)

SHAVIT ET AL.

Examiner

Aimee J. Li

Art Unit

2183

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --
Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 13 February 2006.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1-43 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1-43 is/are rejected.
- 7) ☒ Claim(s) 27,28,30 and 31 is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☐ The drawing(s) filed on _____ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
1. ☐ Certified copies of the priority documents have been received.
 2. ☐ Certified copies of the priority documents have been received in Application No. _____.
 3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- * See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)
Paper No(s)/Mail Date _____
- 4) ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: _____

DETAILED ACTION

1. Claims 1-43 have been considered.
2. In view of the Appeal Brief filed on 13 February 2006, PROSECUTION IS HEREBY REOPENED. The rejection is set forth below.
3. To avoid abandonment of the application, appellant must exercise one of the following two options:

- (1) file a reply under 37 CFR 1.111 (if this Office action is non-final) or a reply under 37 CFR 1.113 (if this Office action is final); or,
- (2) initiate a new appeal by filing a notice of appeal under 37 CFR 41.31 followed by an appeal brief under 37 CFR 41.37. The previously paid notice of appeal fee and appeal brief fee can be applied to the new appeal. If, however, the appeal fees set forth in 37 CFR 41.20 have been increased since they were previously paid, then appellant must pay the difference between the increased fees and the amount previously paid.

Claim Objections

4. Claims 27, 28, 30, and 31 are objected to because of the following informalities: Please correct claims 27, 28, 30, and 31, line 3 in each claim from "...opposing/same ends of the array;" to read --...opposing/same ends of the array[[]];--. Appropriate correction is required.

Claim Rejections - 35 USC § 112

5. The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

6. Claims 27 and 28 recite the limitation "the competing pop operations" in line 2. There is insufficient antecedent basis for this limitation in the claim. Claim 25 does not set the precedent for competing pop operations, just competing operations. The precedent is set in claim 26.

7. Claims 30 and 31 recite the limitation "the competing push operations" in line 2. There is insufficient antecedent basis for this limitation in the claim. Claim 25 does not set the precedent for competing push operations, just competing operations. The precedent is set in claim 29.

Claim Rejections - 35 USC § 101

8. 35 U.S.C. 101 reads as follows:

Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

9. Claims 40-42 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. Claim recites "a computer program product encoded in at least one computer readable medium", which is further narrowed by claim 42 to include wires. Wires, which an embodiment of signals, is non-statutory subject matter. Also, "a computer program product encoded in at least one computer readable medium" includes a program written on a piece of paper.

Claim Rejections - 35 USC § 102

10. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

11. Claims 1-2, 4-5, 15-19, 25-28, and 30-31 are rejected under 35 U.S.C. 102(b) as being taught by Janice M. Stone's "A simple and correct shared-queue algorithm using Compare-and-Swap" ©1990 (herein referred to as Stone).

12. Referring to claim 1, Stone has taught a method of managing access to an array susceptible to concurrent operations on a sequence encoded therein, the method comprising:

- a. Executing as part of a pop operation, an atomic dual target compare and swap (DCAS) to update a then-current, end identifying index for the array and an element of the array adjacent to that identified by the end identifying index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
- b. Returning from the DCAS, on failure thereof, an indication by which an empty state of the array is detectable (Stone Section **4. The Shared-Queue Algorithm** and Figure 5).

13. Referring to claim 2, Stone has taught wherein the indication by which the empty state of the array is detectable is indicative of presence of a distinguishing value in the adjacent element (Stone Section **4. The Shared-Queue Algorithm** and Figure 5).

14. Referring to claim 4, Stone has taught

- a. Wherein the pop operation is a left pop operation (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);
- b. Wherein the end identifying index is a left-end index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
- c. Wherein the adjacent element is to the right of the identified element (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

15. Referring to claim 5, Stone has taught

- a. Wherein the pop operation is a right pop operation (Stone Section 1.
Introduction; Section 2. **Background**, paragraphs 3 and 5-6; Section 3.
Compare-and-Swap; Section 3.1 **The Compare-and-Swap Instruction**,
paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The**
Compare-and-Swap-Double Instruction, paragraphs 1 and 4; Section 4. **The**
Shared-Queue Algorithm; Section 5. **State diagram for the shared queue**,
paragraphs 1-2 and 5; and Figure 5);
- b. Wherein the end identifying index is a right-end index (Stone Section 1.
Introduction; Section 2. **Background**, paragraphs 3 and 5-6; Section 3.
Compare-and-Swap; Section 3.1 **The Compare-and-Swap Instruction**,
paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The**
Compare-and-Swap-Double Instruction, paragraphs 1 and 4; Section 4. **The**
Shared-Queue Algorithm; Section 5. **State diagram for the shared queue**,
paragraphs 1-2 and 5; and Figure 5); and
- c. Wherein the adjacent element is to the left of the identified element (Stone
Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section
3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**,
paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The**
Compare-and-Swap-Double Instruction, paragraphs 1 and 4; Section 4. **The**
Shared-Queue Algorithm; Section 5. **State diagram for the shared queue**,
paragraphs 1-2 and 5; and Figure 5).

16. Referring to claim 15, Stone has taught a method of managing concurrent access to double-ended queue (deque), the method comprising:

- a. Employing, in an implementation of a pop operation, execution of a an atomic dual target compare and swap (DCAS) to interrogate instantaneous values of a first end index and a deque element adjacent to that identified thereby for a signature indicative of an empty state of the array, the signature including presence in that adjacent element of a distinguishing value (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),
- b. Wherein successful execution of an opposing end pop operation includes execution of a DCAS to update a second end index and a deque element adjacent to that identified thereby, the update of that adjacent element storing the distinguishing value therein (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**;

Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

17. Referring to claim 16, Stone has taught wherein successful execution of a competing, same end pop operation includes execution of a DCAS to update the first end index and a deque element adjacent to that identified thereby, the update of that adjacent element storing the distinguishing value therein (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

18. Referring to claim 17, Stone has taught

- a. Wherein the first end index is a left index and, if the state of the deque is non-empty, the deque element adjacent to that identified thereby is a left most element of the deque (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);
- b. Wherein the second end index is a right index and, if the state of the deque is non-empty, the deque element adjacent to that identified thereby is the right most element of the deque (Stone Section 1. **Introduction**; Section 2. **Background**,

paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

19. Referring to claim 18, Stone has taught
 - a. Wherein the pop operation is a left pop operation and the opposing end pop operation is a right pop operation (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
 - b. Wherein the first end index is a left end index and the element adjacent to that identified thereby is adjacent to the right (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

Art Unit: 2183

20. Referring to claim 19, Stone has taught wherein the distinguishing value is encoded as a null value (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

21. Referring to claim 25, Stone has taught a method of managing concurrent access to an array susceptible to competing accesses at same and opposing ends thereof, the method comprising:

- a. Executing as part of a first access operation, an atomic dual target compare and swap (DCAS) to update a first end identifying index and an element of the array corresponding to a then-current value thereof (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);
- b. Executing as part of a competing second access operation, a DCAS to update a second end identifying index and an element of the array corresponding to a then-current value thereof (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The**

Compare-and-Swap Instruction, paragraph 1; **Section 3.2 The A-B-A problem**, paragraphs 2-3; **Section 3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; **Section 4. The Shared-Queue Algorithm**; **Section 5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),

- c. Wherein, if successful completion of one of the first and the second competing access operations results in a boundary condition state of the array, the DCAS of the other of the first and the second access operations fails and returns an indication thereof (Stone **Section 1. Introduction**; **Section 2. Background**, paragraphs 3 and 5-6; **Section 3. Compare-and-Swap**; **Section 3.1 The Compare-and-Swap Instruction**, paragraph 1; **Section 3.2 The A-B-A problem**, paragraphs 2-3; **Section 3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; **Section 4. The Shared-Queue Algorithm**; **Section 5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

22. Referring to claim 26, Stone has taught

- a. Wherein the first access operation and the competing second access operation are competing pop operations (Stone **Section 1. Introduction**; **Section 2. Background**, paragraphs 3 and 5-6; **Section 3. Compare-and-Swap**; **Section 3.1 The Compare-and-Swap Instruction**, paragraph 1; **Section 3.2 The A-B-A problem**, paragraphs 2-3; **Section 3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; **Section 4. The Shared-Queue Algorithm**; **Section 5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

- b. Wherein the array elements corresponding to the first and second indices are each adjacent to that identified by the respective index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);
- c. Wherein the boundary condition state is an empty state (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
- d. Wherein the adjacent element referenced by the failing one of the competing pop operations encodes a distinguishing value signifying the empty state (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The**

Shared-Queue Algorithm; Section 5. State diagram for the shared queue, paragraphs 1-2 and 5; and Figure 5).

23. Referring to claims 27 and 30, Stone has taught
- a. Wherein the competing pop/push operations are competing opposing end pop operations (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
 - b. Wherein the first index and the second index identify opposing ends of the array (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).
24. Referring to claims 28 and 31, Stone has taught
- a. Wherein the competing pop/push operations are competing opposing end pop operations (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3;

Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4;
Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

- b. Wherein the first index and the second index identify same ends of the array
(Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6;
Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3;
Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4;
Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

Claim Rejections - 35 USC § 103

25. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

26. Claim 3 is rejected under 35 U.S.C. 103(a) as being unpatentable over Janice M. Stone's "A simple and correct shared-queue algorithm using Compare-and-Swap" ©1990, as applied to claim 1 above, and further in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999 (herein referred to as Weiss). Stone has taught the end identifying index and an opposing end identifying index delimiting the sequence (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; and Figure 5). Stone has not taught wherein the array encodes a double-ended queue as a circular buffer of bounded size.

Weiss has taught wherein the array encodes a double-ended queue as a circular buffer of bounded size (Weiss page 111). A person of ordinary skill in the art would have recognized, and as taught by Weiss, a circular queue lets elements be added to the beginning of a queue after they have been enqueued, thereby allowing more elements to enter the queue even after all positions have been used at one point. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the circular queue of Weiss in Stone.

27. Claims 6-14, 23-24, and 32-43 are rejected under 35 U.S.C. 103(a) as being unpatentable over Janice M. Stone's "A simple and correct shared-queue algorithm using Compare-and-Swap" ©1990 in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999 (herein referred to as Weiss).

28. Referring to claim 6, Stone has taught a method of managing access to an array susceptible to concurrent operations on a sequence encoded therein, the method comprising executing as part of a push operation, an atomic dual target compare and swap (DCAS) to update a then-current, end identifying index for the array and an element of the array identified by the end identifying index (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5). Stone has not taught returning from an operation, on failure thereof, an indication by which a full state of the array is detectable. Weiss has taught returning from an operation, on failure thereof, an indication by which a full state of the array is detectable (Weiss pages 111-112 and Figure 3.61).

Art Unit: 2183

A person of ordinary skill in the art at the time the invention was made, and as taught by Weiss, that checking whether the array is full or not is part of the push operation to ensure that a nonexistent location is not written to (Weiss page 111), thereby preventing a data memory location error from writing to an invalid array location. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to ensure a data memory location error does not occur.

29. Referring to claim 7, Stone in view of Weiss has taught wherein the indication by which the full state of the array is detectable is indicative of absence of a distinguishing value in the identified element (Weiss pages 111-112 and Figure 3.61).

30. Referring to claim 8, Stone in view of Weiss has taught

- a. Wherein the push operation is a left push operation (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
- b. Wherein the end identifying index is a left-end index (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The**

Shared-Queue Algorithm; Section 5. State diagram for the shared queue,
paragraphs 1-2 and 5; and Figure 5).

31. Referring to claim 9, Stone in view of Weiss has taught
 - a. Wherein the pop operation is a right push operation (Stone Section 1.
Introduction; Section 2. Background, paragraphs 3 and 5-6; Section 3.
Compare-and-Swap; Section 3.1 The Compare-and-Swap Instruction,
paragraph 1; Section 3.2 **The A-B-A problem,** paragraphs 2-3; Section 3.3 **The
Compare-and-Swap-Double Instruction,** paragraphs 1 and 4; Section 4. **The
Shared-Queue Algorithm; Section 5. State diagram for the shared queue,**
paragraphs 1-2 and 5; and Figure 5); and
 - b. Wherein the end identifying index is a right-end index (Stone Section 1.
Introduction; Section 2. Background, paragraphs 3 and 5-6; Section 3.
Compare-and-Swap; Section 3.1 The Compare-and-Swap Instruction,
paragraph 1; Section 3.2 **The A-B-A problem,** paragraphs 2-3; Section 3.3 **The
Compare-and-Swap-Double Instruction,** paragraphs 1 and 4; Section 4. **The
Shared-Queue Algorithm; Section 5. State diagram for the shared queue,**
paragraphs 1-2 and 5; and Figure 5).
32. Referring to claim 10, Stone has taught a method of providing concurrent access to a data structure, the method comprising:
 - a. As part of an access to a first-end of the data structure, performing in alternate
legs of a conditional branch:

- i. A first atomic multi-way compare and swap on then-current contents of a first-end index store and a corresponding element of the double-ended data structure to disambiguate a retry state and a boundary condition state of the double-ended data structure (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);
- ii. A second atomic multi-way compare and swap on then-current contents of the first-end index store and a corresponding element of the double-ended data structure, the second multi-way compare and swap performing the access and, on failure thereof, returning an indication disambiguating a retry state and the boundary condition state of the double-ended data structure (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),

Art Unit: 2183

- b. Wherein the conditional branch discriminates between presence and absence of a distinguishing value in an element of the data structure corresponding to the then-current contents of the first-end index store (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

33. Stone has not taught wherein a double-ended data structure of bounded size implemented using a circular buffer technique. Weiss has taught a double-ended data structure of bounded size implemented using a circular buffer technique (Weiss page 111). A person of ordinary skill in the art would have recognized, and as taught by Weiss, a circular queue lets elements be added to the beginning of a queue after they have been enqueued, thereby allowing more elements to enter the queue even after all positions have been used at one point. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the circular queue of Weiss in Stone.

34. Referring to claim 11, Stone in view of Weiss has taught

- a. Wherein the access includes a pop from the first-end of the double-ended data structure (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3;

Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4;
Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

- b. Wherein the boundary condition state is an empty state of the double-ended data structure (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
- c. Wherein the retry state results from a concurrently performed push or pop access at the first-end of the double-ended data structure (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

35. Referring to claim 12, Stone in view of Weiss has taught

- a. Wherein the access includes a push onto the first-end of the double-ended data structure (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap**

Instruction, paragraph 1; **Section 3.2 The A-B-A problem**, paragraphs 2-3; **Section 3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; **Section 4. The Shared-Queue Algorithm**; **Section 5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

- b. Wherein the boundary condition state is a full state of the double-ended data structure; and
- c. Wherein the retry state results from a concurrently performed push or pop access at the first-end of the double-ended data structure (Stone **Section 1. Introduction**; **Section 2. Background**, paragraphs 3 and 5-6; **Section 3. Compare-and-Swap**; **Section 3.1 The Compare-and-Swap Instruction**, paragraph 1; **Section 3.2 The A-B-A problem**, paragraphs 2-3; **Section 3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; **Section 4. The Shared-Queue Algorithm**; **Section 5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

36. Referring to claim 13, Stone in view of Weiss has taught wherein the double-ended data structure includes a double-ended queue (deque) (Weiss page 111).

37. Referring to claim 14, Stone in view of Weiss has taught wherein the multi-way compare and swap is a dual target compare and swap (DCAS) (Stone **Section 1. Introduction**; **Section 2. Background**, paragraphs 3 and 5-6; **Section 3. Compare-and-Swap**; **Section 3.1 The Compare-and-Swap Instruction**, paragraph 1; **Section 3.2 The A-B-A problem**, paragraphs 2-3; **Section 3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; **Section 4.**

The Shared-Queue Algorithm; Section 5. State diagram for the shared queue, paragraphs 1-2 and 5; and Figure 5).

38. Referring to claim 23, Stone has taught a method of managing concurrent access to double-ended queue (deque), the method comprising:

- a. Employing, in an implementation of a push operation, execution of an atomic dual target compare and swap (DCAS) to interrogate instantaneous values of a third end index and a deque element identified thereby for a signature of the deque, the signature including absence in that identified deque element of a distinguishing value (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),
- b. Wherein successful execution of an opposing and push operation includes execution of a DCAS to update a fourth end index and a deque element identified thereby, the update of the identified deque element storing a value other than the distinguishing value therein (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**;

Section 5. State diagram for the shared queue, paragraphs 1-2 and 5; and Figure 5).

39. Stone has not taught the signature is indicative of a full state of the deque. Weiss has taught the signature is indicative of a full state of the deque (Weiss pages 111-112 and Figure 3.61). A person of ordinary skill in the art at the time the invention was made, and as taught by Weiss, that checking whether the array is full or not is part of the push operation to ensure that a nonexistent location is not written to (Weiss page 111), thereby preventing a data memory location error from writing to an invalid array location. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to ensure a data memory location error does not occur.

40. Referring to claim 24, Stone in view of Weiss has taught wherein successful execution of a competing, same end push operation includes execution of a DCAS to atomically update the first end index and a deque element identified thereby, the update of that adjacent element storing a value other than the distinguishing value therein (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue, paragraphs 1-2 and 5; and Figure 5).**

41. Referring to claim 32, Stone has taught a double-ended queue (deque) implementation comprising a computer readable encoding of at least a first access operation, execution of the first access operation operating at a particular end of the sequence and employing an atomic dual

target compare and swap (DCAS) to update a corresponding one, but not both, of the left and right indices L and R and an element of the contiguous array adjacent to the contiguous array element identified thereby (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5). Stone has not taught

- a. A contiguous array S of bounded size encoded in an addressable store;
- b. A left index L and a right index R into the contiguous array, the contiguous array S, the left index L and the right index R together defining a circular buffer with state including a sequence of zero or more values encoded in the contiguous array between elements S[L] and S[R] thereof and wherein at least the S[L] and S[R] entries encode a distinguishing value.

42. Weiss has taught

- a. A contiguous array S of bounded size encoded in an addressable store (Weiss page 111);
- b. A left index L and a right index R into the contiguous array, the contiguous array S, the left index L and the right index R together defining a circular buffer with state including a sequence of zero or more values encoded in the contiguous array between elements S[L] and S[R] thereof and wherein at least the S[L] and S[R] entries encode a distinguishing value (Weiss page 111).

Art Unit: 2183

43. A person of ordinary skill in the art would have recognized, and as taught by Weiss, a circular queue lets elements be added to the beginning of a queue after they have been enqueued, thereby allowing more elements to enter the queue even after all positions have been used at one point. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the circular queue of Weiss in Stone.

44. Referring to claim 33, Stone in view of Weiss has taught

- a. Wherein the first access operation includes a push (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
- b. Wherein, on failure, the DCAS returns an indication by which a full state of the contiguous array is detected (Weiss pages 111-112 and Figure 3.61).

45. Referring to claim 34, Stone in view of Weiss has taught

- a. Wherein the first access operation includes a pop (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**;

Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

- b. Wherein, on failure, the DCAS returns an indication by which an empty state of the contiguous array is detected (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

46. Referring to claim 35, Stone in view of Weiss has taught

- a. Computer readable encodings of at least three additional access operations (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),
- b. Wherein the first and three additional access operations include push and pop operations at left and rights end of the sequence, respectively (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**,

paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

47. Referring to claim 36, Stone has taught a concurrent shared object implementation comprising:

- a. A computer readable encoding of push and pop operations defined to operate on elements of the contiguous array and on respective of the opposing indices (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),
- b. Wherein the push operation employs a first instance of an atomic dual target compare and swap (DCAS) operation to update one of the opposing indices and a corresponding element of the contiguous array (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**;

Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5), and

- c. Wherein the pop operation employs a second instance of a DCAS operation to update one of the opposing indices and a corresponding element of the contiguous array while returning on failure, an indication by which an empty state of the contiguous array is detected (Stone Section 1. **Introduction**; Section 2.

Background, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

- 48. Stone has not taught
 - a. A contiguous array encoded in an addressable store;
 - b. Opposing indices into the contiguous array usable to delimit therebetween a portion of the contiguous array for storage of a sequence of zero or more data values; and
 - c. Returning on failure, an indication by which a full state of the contiguous array is detected.
- 49. Weiss has taught
 - a. A contiguous array encoded in an addressable store (Weiss pages 111-112 and Figure 3.61);

- b. Opposing indices into the contiguous array usable to delimit therebetween a portion of the contiguous array for storage of a sequence of zero or more data values (Weiss pages 111-112 and Figure 3.61); and
- c. Returning on failure, an indication by which a full state of the contiguous array is detected (Weiss pages 111-112 and Figure 3.61).

50. A person of ordinary skill in the art at the time the invention was made, and as taught by Weiss, that checking whether the array is full or not is part of the push operation to ensure that a nonexistent location is not written to (Weiss page 111), thereby preventing a data memory location error from writing to an invalid array location. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to ensure a data memory location error does not occur.

51. Referring to claim 37, Stone in view of Weiss has taught

- a. Wherein concurrent shared object includes a deque (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
- b. Wherein the computer readable encoding of push and pop operations includes (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap**

- Instruction**, paragraph 1; **Section 3.2 The A-B-A problem**, paragraphs 2-3; **Section 3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; **Section 4. The Shared-Queue Algorithm**; **Section 5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);
- c. Opposing end variants of the pop operation (Stone **Section 1. Introduction**; **Section 2. Background**, paragraphs 3 and 5-6; **Section 3. Compare-and-Swap**; **Section 3.1 The Compare-and-Swap Instruction**, paragraph 1; **Section 3.2 The A-B-A problem**, paragraphs 2-3; **Section 3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; **Section 4. The Shared-Queue Algorithm**; **Section 5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
- d. Opposing end variants of the push operation (Stone **Section 1. Introduction**; **Section 2. Background**, paragraphs 3 and 5-6; **Section 3. Compare-and-Swap**; **Section 3.1 The Compare-and-Swap Instruction**, paragraph 1; **Section 3.2 The A-B-A problem**, paragraphs 2-3; **Section 3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; **Section 4. The Shared-Queue Algorithm**; **Section 5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).
52. Referring to claim 38, Stone in view of Weiss has taught
- a. Wherein concurrent shared object includes a queue or FIFO (Stone **Section 1. Introduction**; **Section 2. Background**, paragraphs 3 and 5-6; **Section 3. Compare-and-Swap**; **Section 3.1 The Compare-and-Swap Instruction**,

paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

- b. Wherein the computer readable encoding of push and pop operations operate on opposing ends of the queue or FIFO (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

53. Referring to claim 39, Stone in view of Weiss has taught

- a. Wherein concurrent shared object includes a stack or LIFO (Weiss page 93); and
- b. Wherein the computer readable encoding of push and pop operations operate on a same end of the stack or LIFO (Weiss page 93).

54. Referring to claim 40, Stone has taught a computer program product encoded in at least one computer readable medium, the computer program product comprising:

- a. Instances of the at least one functional sequence concurrently executable by plural processors of a multiprocessor and each including an atomic dual target compare and swap (DCAS) to update a corresponding one of the end identifying indices and an element of the array corresponding to a then-current value thereof (Stone

Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

- b. The DCAS of the at least one functional sequence responsive to a corresponding boundary condition state of the concurrent shared object (Stone Section 1.

Introduction; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

55. Stone has not taught at least one functional sequence implementing an access operation on a concurrent shared object, the concurrent shared object instantiable as a circular buffer of bounded size implementing a contiguous array delimited by a pair of end identifying indices.

Weiss has taught at least one functional sequence implementing an access operation on a concurrent shared object, the concurrent shared object instantiable as a circular buffer of bounded size implementing a contiguous array delimited by a pair of end identifying indices (Weiss pages 111-112 and Figure 3.61). A person of ordinary skill in the art would have recognized, and as taught by Weiss, a circular queue lets elements be added to the beginning of a

Art Unit: 2183

queue after they have been enqueued, thereby allowing more elements to enter the queue even after all positions have been used at one point. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the circular queue of Weiss in Stone.

56. Referring to claim 41, Stone in view of Weiss has taught

- a. Wherein the at least one functional sequence includes opposing end variants of push and pop operations on the concurrent shared object (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);
- b. Wherein the boundary condition state corresponding to push operations is a full state of the array (Weiss pages 111-112 and Figure 3.61); and
- c. Wherein the boundary condition state corresponding to pop operations is an empty state of the array (Stone Section 1. **Introduction**; Section 2. **Background**, paragraphs 3 and 5-6; Section 3. **Compare-and-Swap**; Section 3.1 **The Compare-and-Swap Instruction**, paragraph 1; Section 3.2 **The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section 4. **The Shared-Queue Algorithm**; Section 5. **State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

57. Referring to claim 42, Stone in view of Weiss has taught wherein the at least one computer readable medium is selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium and a network, wireline, wireless or other communications medium (Stone Abstract; Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

58. Referring to claim 43, Stone has taught an apparatus comprising:

- a. Plural processors (Stone Abstract; Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);
- b. A store addressable by each of the plural processors (Stone Abstract; Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The**

- Shared-Queue Algorithm; Section 5. State diagram for the shared queue, paragraphs 1-2 and 5; and Figure 5);**
- c. Means for coordinating competing access operations, the coordinating means employing in each instance thereof, at least one atomic dual target compare and swap (DCAS) operation to disambiguate a retry state and a boundary condition state of the array based on then-current contents of one, but not both, of first- and second-end index stores and an array element corresponding thereto (Stone Abstract; **Section 1. Introduction; Section 2. Background, paragraphs 3 and 5-6; Section 3. Compare-and-Swap; Section 3.1 The Compare-and-Swap Instruction, paragraph 1; Section 3.2 The A-B-A problem, paragraphs 2-3; Section 3.3 The Compare-and-Swap-Double Instruction, paragraphs 1 and 4; Section 4. The Shared-Queue Algorithm; Section 5. State diagram for the shared queue, paragraphs 1-2 and 5; and Figure 5).**
59. Stone has not taught first- and second-end index stores accessible to each of the plural processors for identifying opposing ends of a bounded-size contiguous array encoded in circular buffer form in the addressable store. Weiss has taught first- and second-end index stores accessible to each of the plural processors for identifying opposing ends of a bounded-size contiguous array encoded in circular buffer form in the addressable store (Weiss pages 111-112 and Figure 3.61). A person of ordinary skill in the art would have recognized, and as taught by Weiss, a circular queue lets elements be added to the beginning of a queue after they have been enqueued, thereby allowing more elements to enter the queue even after all positions have been

used at one point. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the circular queue of Weiss in Stone.

60. Claims 20-22 are rejected under 35 U.S.C. 103(a) as being unpatentable over Janice M. Stone's "A simple and correct shared-queue algorithm using Compare-and-Swap" ©1990, as applied to claim 15 above, and further in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999 (herein referred to as Weiss).

61. Referring to claim 20, Stone has taught

- a. Employing, in an implementation of a push operation, execution of an atomic dual target compare and swap (DCAS) to interrogate instantaneous values of a third end index and a deque element identified thereby for a signature of the deque, the signature including absence in that identified deque element of a distinguishing value (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),
- b. Wherein successful execution of an opposing and push operation includes execution of a DCAS to update a fourth end index and a deque element identified thereby, the update of the identified deque element storing a value other than the distinguishing value therein (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1**

The Compare-and-Swap Instruction, paragraph 1; **Section 3.2 The A-B-A problem**, paragraphs 2-3; **Section 3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; **Section 4. The Shared-Queue Algorithm**; **Section 5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

62. Stone has not taught the signature is indicative of a full state of the deque. Weiss has taught the signature is indicative of a full state of the deque (Weiss pages 111-112 and Figure 3.61). A person of ordinary skill in the art at the time the invention was made, and as taught by Weiss, that checking whether the array is full or not is part of the push operation to ensure that a nonexistent location is not written to (Weiss page 111), thereby preventing a data memory location error from writing to an invalid array location. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to ensure a data memory location error does not occur.

63. Referring to claim 21, Stone in view of Weiss has taught

- a. Wherein the first end index and the third end index identify a same end of the deque (Stone **Section 1. Introduction**; **Section 2. Background**, paragraphs 3 and 5-6; **Section 3. Compare-and-Swap**; **Section 3.1 The Compare-and-Swap Instruction**, paragraph 1; **Section 3.2 The A-B-A problem**, paragraphs 2-3; **Section 3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; **Section 4. The Shared-Queue Algorithm**; **Section 5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

- b. Wherein the second end index and the fourth end index identify a same end of the deque (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).
64. Referring to claim 22, Stone in view of Weiss has taught
- a. Wherein the first end index and the fourth end index identify a same end of the deque (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
 - b. Wherein the second end index and the third end index identify a same end of the deque (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

Art Unit: 2183

65. Claim 29 is rejected under 35 U.S.C. 103(a) as being unpatentable over Janice M. Stone's "A simple and correct shared-queue algorithm using Compare-and-Swap" ©1990, as applied to claim 25 above, and further in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999 (herein referred to as Weiss). Stone has taught

- a. Wherein the first access operation and the competing second access operation are competing push operations (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);
- b. Wherein the array elements corresponding-to the first and second indices are each identified by the respective index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and
- c. Wherein the array element referenced by the failing one of the competing push operations encodes a value other than a distinguishing value (Stone Section **1.**

Introduction; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section **3.3 The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

66. Stone has not taught wherein the boundary condition state is a full state. Weiss has taught wherein the boundary condition state is a full state (Weiss pages 111-112 and Figure 3.61). A person of ordinary skill in the art at the time the invention was made, and as taught by Weiss, that checking whether the array is full or not is part of the push operation to ensure that a nonexistent location is not written to (Weiss page 111), thereby preventing a data memory location error from writing to an invalid array location. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to ensure a data memory location error does not occur.

Conclusion

67. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure as follows. Applicant is reminded that in amending in response to a rejection of claims, the patentable novelty must be clearly shown in view of the state of the art disclosed by the references cited and the objections made. Applicant must also show how the amendments avoid such references and objections. See 37 CFR § 1.111(c).

Art Unit: 2183

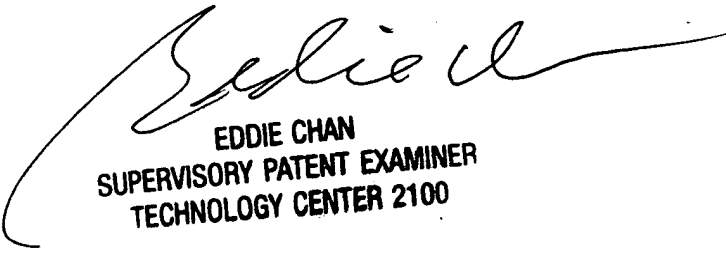
- a. Maged M. Michael and Michael L. Scott's "Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms" ©1996 teaches a method for managing concurrent shared array accesses.

68. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Aimee J. Li whose telephone number is (571) 272-4169. The examiner can normally be reached on M-T 7:00am-4:30pm.

69. If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

70. Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

AJL
Aimee J. Li
27 April 2006



EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100